

Towards single face shortest vertex-disjoint paths in undirected planar graphs

Glencora Borradaile, Amir Nayyeri, and Farzad Zafarani

School of Electrical Engineering and Computer Science
Oregon State University
{glencora, nayyeri, zafaranf}@eecs.oregonstate.edu

July 23, 2015

Abstract

Given k pairs of terminals $\{(s_1, t_1), \dots, (s_k, t_k)\}$ in a graph G , the min-sum k vertex-disjoint paths problem is to find a collection $\{Q_1, Q_2, \dots, Q_k\}$ of vertex-disjoint paths with minimum total length, where Q_i is an s_i -to- t_i path between s_i and t_i . We consider the problem in planar graphs, where little is known about computational tractability, even in restricted cases. Kobayashi and Sommer propose a polynomial-time algorithm for $k \leq 3$ in undirected planar graphs assuming all terminals are adjacent to at most two faces. Colin de Verdière and Schrijver give a polynomial-time algorithm when all the sources are on the boundary of one face and all the sinks are on the boundary of another face and ask about the existence of a polynomial-time algorithm provided all terminals are on a common face.

We make progress toward Colin de Verdière and Schrijver's open question by giving an $O(kn^5)$ time algorithm for undirected planar graphs when $\{(s_1, t_1), \dots, (s_k, t_k)\}$ are in counter-clockwise order on a common face.

1 Introduction

Given k pairs of terminals $\{(s_1, t_1), \dots, (s_k, t_k)\}$, the k *vertex-disjoint* paths problem asks for a set of k disjoint paths $\{Q_1, Q_2, \dots, Q_k\}$, in which Q_i is a path between s_i and t_i for all $1 \leq i \leq k$. As a special case of the multi-commodity flow problem, computing vertex disjoint paths has found several applications, for example in VLSI design [KvL84], or network routing [ORS93, SM05]. It is one of Karp’s NP-hard problems [Kar74] even for undirected planar graphs if k is part of the input [MP93]. However, there are polynomial time algorithms if k is a constant for general undirected graphs [RS95, KW10]. In general directed graphs, the k -vertex-disjoint paths problem is NP-hard even for $k = 2$ [FHW80] but is fixed parameter tractable with respect to parameter k in directed planar graphs [Sch94, CMPP13].

Surprisingly, much less is known for the optimization variant of the problem, *minimum-sum k vertex-disjoint paths problem* (k -min-sum), where a set of disjoint paths with *minimum total length* is desired. For example, the 2-min-sum problem and the 4-min-sum problem are open in directed and undirected planar graphs, respectively, even when the terminals are on a common face; neither polynomial-time algorithms nor hardness results are known for these problems [KS10]. Bjorklund and Husfeldt gave a randomized polynomial time algorithm for the min-sum two vertex-disjoint paths problem in general undirected graphs [BH14]. Kobayashi and Sommer provide a comprehensive list of similar open problems (Table 2 [KS10]).

One of a few results in this context is due to Colin de Verdière and Schrijver [VS11]: a polynomial time algorithm for the k -min-sum problem in a (directed or undirected) planar graph, given all sources are on one face and all sinks are on another face [VS11]. In the same paper, they ask about the existence of a polynomial time algorithm provided all the terminals (sources and sinks) are on a common face. If the sources and sinks are ordered so that they are in the order $s_1, s_2, \dots, s_k, t_k, t_{k-1}, \dots, t_1$ around the boundary, then the k -min-sum problem can be solved by finding a min-cost flow from s_1, s_2, \dots, s_k to t_k, t_{k-1}, \dots, t_1 . For $k \leq 3$ in undirected planar graphs with the terminals in arbitrary order around the common face, Kobayashi and Sommer give an $O(n^4 \log n)$ algorithm [KS10]¹. In this paper, we give the first polynomial-time algorithm for an arbitrary number of terminals on the boundary of a common face, which we call F , so long as the terminals alternate along the boundary. Formally, we prove:

Theorem 1.1. *There exists an $O(kn^5)$ time algorithm to solve the k -min-sum problem, provided that the terminals $s_1, t_1, s_2, t_2, \dots, s_k, t_k$ are in counter-clockwise order on the boundary of the graph.*

Definitions and assumptions We use standard notation for graphs and planar graphs. For simplicity, we assume that the terminal vertices are distinct. One could imagine allowing $t_i = s_{i+1}$; our algorithm can be easily modified to handle this case. We also assume that the shortest path between any two vertices of the input graph is unique as it significantly simplifies the presentation of our result; this assumption can be enforced using a perturbation technique [MVV87].

2 Preliminaries

Walks and paths. Let $G = (V, E)$ be a graph, and let H be a subgraph of G . We use $V(H)$ and $E(H)$ to denote the vertex set and the edge set of H , respectively. For $U \subseteq V$, we use $G[U]$ to denote the subgraph of G induced by U , whose vertex set is U and whose edge set is all edges of G with both endpoints in U . A *walk* $W = (v_1, v_2, \dots, v_h)$ in G is a sequence of vertices such

¹Kobayashi and Sommer also describe algorithms for the case where terminals are on two different faces, and $k = 3$.

that for all $1 \leq i < h$, we have $(v_i, v_{i+1}) \in E$. For any $v_i, v_j \in W$, $W[v_i, v_j]$ is the (sub-)walk $(v_i, v_{i+1}, \dots, v_j)$. A *path* is a walk with no repeated vertices. Sometimes, we view a walk as its set of edges, and use set operations on walks. For example, given two walks W_1 and W_2 , we define $W = W_1 \oplus W_2$ to be their symmetric difference when it is clear from the context that W is a walk. Given a length function $\ell : E \rightarrow V$, the length of W is denoted by $\ell(W)$, and it is $\sum_{i=1}^{h-1} \ell(v_i, v_{i+1})$.

Planarity. An *embedding* of a graph $G = (V, E)$ into the Euclidean plane is a mapping of vertices of G into different points of \mathbb{R}^2 , and edges of G into internally disjoint simple curves in \mathbb{R}^2 such that the endpoints of the image of $(u, v) \in E$ are the images of vertices $u \in V$ and $v \in V$. If such an embedding exists then G is a *planar graph*. A *plane graph* is a planar graph and an embedding of it. The *faces* of a plane graph G are the maximal connected components of \mathbb{R}^2 that are disjoint from the image of G . If G is connected it contains only one unbounded face. This is called the *outer face* of G and we denote the boundary of G by ∂G . Let W be a walk and let W^o be the set of edges that are used an odd number of times in W . W^o is a collection of simple cycles. For any point $p \in \mathbb{R}^2$, we say that p is inside W if p is on the image of W in the plane, or p is inside an odd number of cycles of W^o . When it is clear from the context, we use the same notation to refer to the subgraph composed of the vertices and edges whose images are completely inside W .

3 Structural properties

In this section, we present fundamental properties of the optimum solution that we exploit in our algorithm. To simplify the exposition, we search for pairwise disjoint walks rather than simple paths and refer to a set of pairwise disjoint walks connecting corresponding pair of terminals as a *feasible* solution. Indeed, in an optimal solution, the walks are simple paths.

Let $\{Q_1, Q_2, \dots, Q_k\}$ be an optimal solution, where Q_i is a s_i -to- t_i path and let $\{P_1, P_2, \dots, P_k\}$ be the set of shortest paths, where P_i is the s_i -to- t_i shortest path. These shortest paths together with the boundary of the graph, ∂G , define internally disjoint regions of the plane. Specifically, we define R_i to be the subset of the plane bounded by the cycle $P_i \cup C_i$, where C_i is the s_i -to- t_i subpath of ∂G that does not contain other terminal vertices. The following lemmas constrain the behavior of the optimal paths.

Lemma 3.1. *For all $1 \leq i \leq k$, the path Q_i is inside R_i .*

Proof. Suppose, to derive a contradiction, that $Q_i \not\subseteq R_i$. So, there is a vertex $v \in Q_i \setminus R_i$. Let p_v be the maximal path containing v that is internally disjoint from R_i . Let (x, y) be endpoints of p_v , and observe that $x, y \in P_i$. Also, by uniqueness of shortest paths, we have $\ell(P_i[x, y]) < \ell(Q_i[x, y])$. Thus, $Q'_i = Q_i \oplus Q_i[x, y] \oplus P_i[x, y]$ is shorter than Q_i .

It remains to show that $Q'_i \cap Q_j = \emptyset$ for all $j \neq i$, $1 \leq j \leq k$. But, by the construction, Q'_i is inside $C_i \cup Q_i$, and all terminals other than s_i and t_i are outside $C_i \cup Q_i$. So, by Jordan curve theorem, for any Q_j to intersect Q'_i , it has to intersect Q_i , too. Thus, $\{Q_1, \dots, Q_k\} \setminus Q_i \cup Q'_i$ is a shorter solution than the optimum, which is a contradiction. \square

We take the vertices of P_i and Q_i to be ordered along these paths from s_i to t_i .

Lemma 3.2. *For $u, v \in Q_i \cap P_i$, u precedes v in P_i if and only if u precedes v in Q_i .*

Proof. Suppose, to derive a contradiction, that u and v have different orders on $P_i[s_i, t_i]$ and $Q_i[s_i, t_i]$, and assume, without loss of generality, that u precedes v in $P_i[s_i, t_i]$, but v precedes u in $Q_i[s_i, t_i]$. So, $Q_i[s_i, t_i]$ can be decomposed into three subpaths (1) γ_1 is a s_i -to- v path, (2) γ_2 is a

v -to- u path, and (3) γ_3 is a u -to- t_i path. If γ_1 contains u then Q_i is not a simple path, visiting u at least twice. Otherwise, The Jordan curve theorem implies that γ_2 or γ_3 must intersect γ_1 . Again Q_i is not simple, so, it is not a walk in the optimum solution. \square

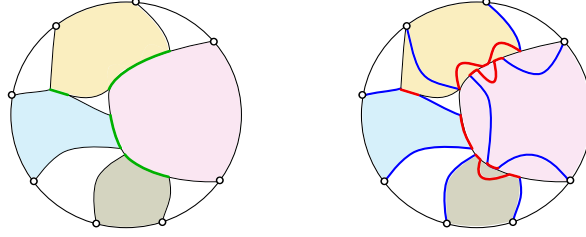


Figure 1: (left) A 4-min sum instance; regions are shaded and borders are green. (right) A feasible solution; Type I and Type II subpaths are blue and red, respectively.

We call $R_i \cap R_j$ the *border* of R_i and R_j and denote it $B_{i,j}$. Note that a border can be a single vertex. Since we assume shortest paths are unique, $B_{i,j}$ is a single (shortest) path. Figure 1 illustrates borders for a 4-min-sum instance. The following lemma bounds the total number of borders.

Lemma 3.3. *There are $O(k)$ border paths.*

Proof. Let $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ be the graph whose vertices correspond to regions in G , and there is an edge between two vertices of $V_{\mathcal{R}}$ if their corresponding regions in G share a border. Let $H = G[\partial F \cup P_1 \cup \dots \cup P_k]$, and observe that \mathcal{R} is a subgraph of the planar dual of H . Thus, \mathcal{R} is planar.

Since there is a bijection between $V_{\mathcal{R}}$ and the set of regions of G , we have $|V_{\mathcal{R}}| = k$. Additionally, there is a bijection between $E_{\mathcal{R}}$ and borders in G . Because, \mathcal{R} is planar we conclude that the number of borders in G is $|E_{\mathcal{R}}| = O(V_{\mathcal{R}}) = O(k)$. \square

Consider a region R_i and consider the borders along $P_i, B_{i,i_1}, B_{i,i_2}, \dots, B_{i,i_t}$. Observe that the intersections of the regions $R_{i_1}, R_{i_2}, \dots, R_{i_t}$ with ∂G must be in a *clockwise* order. Let $\iota_1, \dots, \iota_\ell$ be the subsequence of i_1, \dots, i_t of indices to regions that intersect Q_i . For $j \in \{\iota_1, \dots, \iota_\ell\}$, let x_j and y_j be the first and last vertex of Q_i in $B_{i,j}$. Additionally, define $y_0 = s_i$ and $x_{\ell+1} = t_i$. We partition Q_i into a collection of subpaths of two types as follows.

Type I : For $h = 0, \dots, \ell$, $Q_i[y_h, x_{h+1}]$ is a *Type I* subpath in region R_i .

Type II : For $h = 1, \dots, \ell - 1$, $Q_i[x_h, y_h]$ is a *Type II* subpath in region R_i . We say that $Q_i[x_h, y_h]$ is on the border $B_{i,j}$ containing x_h and y_h .

By this definition, all Type I paths are internally disjoint from all borders. By Lemma 3.2, each Type II path is internally disjoint from all borders except possibly the border that contains its endpoints, with which it may have several intersecting points. See Figure 1 for an illustration of Type I and II paths.

The following lemma demonstrates a key property of Type I paths, implying that (given their endpoints) they can be computed efficiently via a shortest path computation:

Lemma 3.4. *Let α be a Type I subpath in region R_i . Then α is the shortest path between its endpoints in R_i that is internally disjoint from all borders.*

Proof. Let $u, v \in V(G)$ be the endpoints of α , and let α' be the shortest u -to- v path in R_i that is internally disjoint from all borders. Also, let $Q'_i = Q_i \oplus \alpha \oplus \alpha'$. The path α' has the same endpoints as α , and it is internally disjoint from all R_j if $j \neq i$. Also, by Lemma 3.1, for each $1 \leq j \leq k$, Q_j is inside R_j . Therefore, α' is disjoint from Q_j for all $1 \leq j \leq k$ and $j \neq i$. Thus, $\{Q_1, \dots, Q_k\} \setminus Q_i \cup Q'_i$ is a set of k pairwise disjoint walks, with total length $OPT - \ell(\alpha) + \ell(\alpha')$ where OPT is the total length of the optimal paths. So, $OPT - \ell(\alpha) + \ell(\alpha') \geq OPT$, which implies $\ell(\alpha') \geq \ell(\alpha)$. Therefore α must be a shortest (u, v) path in R_i that avoids boundaries. In fact, uniqueness of shortest paths implies $\alpha = \alpha'$. \square

A Type II path has a similar property if it is the only Type II path on the border that contains its endpoints. The proof of the following lemma is almost exactly the same as the previous proof.

Lemma 3.5. *Let β be a Type II subpath in region R_i on border $B_{i,j}$. Suppose there is no Type II path on $B_{i,j}$ inside R_j . Then β is the subpath of $B_{i,j}$ between its endpoints.*

The following lemma reveals a relatively more sophisticated structural property of Type II paths on shared borders.

Lemma 3.6. *Let β and γ be Type II subpaths in R_i and R_j on $B_{i,j}$, respectively, let x_i and y_i be the endpoints of β , and let x_j and y_j be the endpoints of γ . Then, $\{\beta, \gamma\}$ is the pair of paths with minimum total length with the following properties:*

- (1) β is an x_i -to- y_i path inside R_i , and it is internally disjoint from all borders except possibly $B_{i,j}[x_i, y_i]$.
- (2) γ is an x_j -to- y_j path inside R_j , and it is internally disjoint from all borders except possibly $B_{i,j}[x_j, y_j]$.

Proof. Properties (1) and (2) of β and γ are implied by the definition of Type II paths and because they are internally disjoint from all borders except $B_{i,j}$. It remains to show that their total length is minimum.

Let (β', γ') be the pair of paths with minimum total length that has properties (1) and (2). Let $Q'_i = Q_i \oplus \beta \oplus \beta'$, and let $Q'_j = Q_j \oplus \gamma \oplus \gamma'$.

By construction, β' is internally disjoint from all borders except (possibly) $B_{i,j}[x_i, y_i]$. Additionally, the endpoints of β' are the same as the endpoints of β . Therefore, intersection points of β' with borders of R_i that are not in β are all in $B_{i,j}[x_i, y_i]$. Similarly, intersection points of γ' with borders of R_j that are not in γ are all on $B_{i,j}[x_j, y_j]$. It immediately follows that Q'_i and Q_h are disjoint for any $h \in \{1, 2, \dots, k\} \setminus \{j\}$. Similarly, Q'_j and Q_h are disjoint for any $h \in \{1, 2, \dots, k\} \setminus \{i\}$.

Furthermore, Q'_i and Q'_j are disjoint by their construction. Thus, $\{Q_1, \dots, Q_k\} \setminus \{Q_i, Q_j\} \cup \{Q'_i, Q'_j\}$ is a set of k pairwise disjoint walks connecting the terminals. The total length of this new set is $OPT - \ell(\beta) - \ell(\gamma) + \ell(\beta') + \ell(\gamma') \geq OPT$. Consequently, $\ell(\beta) + \ell(\gamma) \leq \ell(\beta') + \ell(\gamma')$, in fact, $\ell(\beta) + \ell(\gamma) = \ell(\beta') + \ell(\gamma')$. Thus, (α, β) are a pair of path with minimum total length that has properties (1) and (2). \square

4 Algorithmic toolbox

In this section, we describe algorithms to compute paths of Type I and II for given endpoints. These algorithms are key ingredients of our strongly polynomial time algorithm described in the next section. More directly, they imply an $n^{O(k)}$ time algorithm via enumerating the endpoints, which is sketched at the end of this section.

Each Type I path can be computed in linear time using the algorithms of Henzinger et al. [HKRS97]; they can also be computed in bulk in $O(n \log n)$ time using the multiple-source shortest path algorithm of Klein [Kle05] (although other parts of our algorithms dominate the shortest path computation). Similarly, a Type II path on a border $B_{i,j}$ can be computed in linear time provided it is the only path on $B_{i,j}$. Computing pairs of Type II paths on a shared border is slightly more challenging. To achieve this, our algorithm reduces this problem into a 2-min sum problem that can be solved in linear time via a reduction to the minimum-cost flow problem. The following lemma is implicit in the paper of Kobayashi and Sommer [KS10].

Lemma 4.1. *There exists a linear time algorithm to solve the 2-min sum problem on an undirected planar graph, provided the terminals are on the outer face.*

Proof. Consider an instance of the 2-min sum problem with terminals $\{(s_1, t_1), (s_2, t_2)\}$. This problem reduces to a minimum-cost flow problem as follows. Because of the symmetry for terminals in undirected graphs, we can assume that s_1 and s_2 are next to each other on the outer face: there is a s_1 -to- s_2 path on the boundary of the outer face that does not intersect $\{t_1, t_2\}$. We make the graph directed by replacing each undirected edge $\{u, v\}$ with edges (u, v) and (v, u) . For edge (u, v) in the directed graph, we assign its length using length function $\ell(u, v)$. For every vertex v in G , we split it into two vertices v_1 and v_2 and connect them with a zero length edge that has unit capacity. For each edge (u, v) , we connect u to v_1 , and for each edge (v, u) , we connect v_2 to u . We introduce a dummy source vertex d_1 , with two edges (d_1, s_1) , and (d_1, s_2) of unit capacity and zero length. Also, we introduce a dummy sink vertex d_2 , with edges (t_1, d_2) and (t_2, d_2) of unit capacity and zero length. We assign capacity one to all other edges. The lengths of the other edges are specified by the length function ℓ of G . Since, s_1 and s_2 (also t_1 and t_2) are next to each other, the graph remains planar after adding u , v , and their incident edges. Now, it is straight forward to see that our 2-min sum problem is equivalent to a minimum cost u -to- v flow problem of value 2. This minimum cost flow problem in turn reduces to two shortest path computations from u to v , which can be done in linear time [HKRS97]. \square

We reduce the computation of Type II paths to 2-min sum. The following lemma is a slightly stronger form of this reduction, which finds application in our strongly polynomial time algorithm.

Lemma 4.2. *Let R_i and R_j be two regions with border $B_{i,j}$ and let $x_i, y_i \in P_i$ and $x_j, y_j \in P_j$. A pair of paths (β, γ) with total minimum length and with the following properties can be computed in linear time.*

1. β is an x_i -to- y_i path inside R_i , and it is internally disjoint from all borders except possibly $P_i[x_i, y_i] \cap B_{i,j}$.
2. γ is an x_j -to- y_j path inside R_j , and it is internally disjoint from all borders except possibly $P_j[x_j, y_j] \cap B_{i,j}$.

Proof. Let the graph H be the induced subgraph by the vertices of G inside $R_i \cup R_j$. We obtain H' from H by performing the following operations:

1. deleting all vertices of H that belong to borders other than $B_{i,j}$,
2. deleting all edges in R_i that are incident to $B_{i,j}$ but not incident to $P_i[x_i, y_i] \cap B_{i,j}$, and
3. deleting all edges in R_j that are incident to $B_{i,j}$ but not incident to $P_j[x_j, y_j] \cap B_{i,j}$.

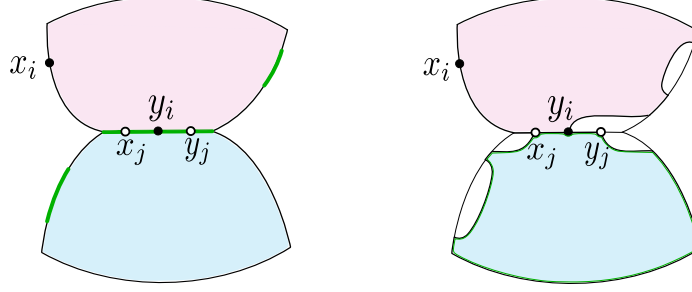


Figure 2: (left) The graph H induced by vertices in $R_i \cup R_j$; regions are shaded and borders are green. (right) The graph H' defined in the proof of Lemma 4.2

Note that H' is not necessarily connected. For an illustration of H and H' , see Figure 2.

Since β and γ are intact in H' , they can be computed in H' instead of G . Furthermore, observe that x_i, y_i, x_j, y_j are on the boundary of H' . If H' is disconnected, then β and γ are a shortest paths in their own connected components. So they can be computed in linear time using the algorithm of Henzinger et al. [HKRS97].

So, suppose H' is connected, and observe that x_i, y_i, x_j, y_j are on the boundary of the outer face of H' . By Lemma 4.1, there is a linear time algorithm to compute a pair of disjoint paths of minimum total length between corresponding terminals.

Let β' and γ' be x_i -to- y_i path and x_j -to- y_j path computed via solving a 2-min sum problem. It remains to prove that β' and γ' have properties (1) and (2) of the lemma. That is $\beta' \in H' \cap R_i$, and $\gamma' \in H' \cap R_j$. This can be done through a replacing path argument similar to Lemma 3.1, as $B_{i,j}$ (so, any subpath of it) is a shortest path. \square

4.1 An $n^{O(k)}$ time algorithm

The properties of Type I and II paths imply a naïve $n^{O(k)}$ time algorithm, which we sketch here. An optimal solution defines the endpoints of Type I and Type II paths, so we can simply enumerate over which borders contain endpoints of Type I and II paths and then enumerate over the choices of the endpoints. Consequently, there are zero, two, or four (not necessarily distinct) endpoints of Type I and II paths on $B_{i,j}$, or

$$1 + \binom{\ell(B_{i,j})}{2} + \binom{\ell(B_{i,j})}{4}$$

possibilities, which is $O(n^4)$ since $\ell(B_{i,j}) = O(n)$. Since there are $O(k)$ borders (Lemma 3.3), there are $n^{O(k)}$ endpoints to guess. Given the set of endpoints, we compute a feasible solution composed of the described Type I and II paths or determines that no such solution exists. Since Type I and II paths can be computed in polynomial time, the overall algorithm runs in $n^{O(k)}$ time.

5 A fully polynomial time algorithm

We give an $O(kn^5)$ -time algorithm via dynamic programming over the regions. For two regions R_i and R_j that have a shared border $B_{i,j}$, R_i and R_j separate the terminal pairs into two sets: those terminals s_ℓ, t_ℓ for $\ell = i + 1, \dots, j - 1$ and s_m, t_m for $m = j + 1, \dots, k, 1, \dots, i - 1$ (for $i < j$). Any s_ℓ -to- t_ℓ path that is in region R_ℓ cannot touch any s_m -to- t_m path that is in region R_m since R_ℓ and R_m are vertex disjoint. Therefore any influence the s_ℓ -to- t_ℓ path has on the s_m -to- t_m path occurs indirectly through the s_i -to- t_i and s_j -to- t_j paths. Our dynamic program is indexed by the shared

borders $B_{i,j}$ and pairs of vertices on (a subpath of) P_i and (a subpath of) P_j ; the vertices on P_i and P_j will indicate a *last point* on the boundary of R_i and R_j that a (partial) feasible solution uses.

We use a tree to order the shared borders for processing by the dynamic program. Since there are $O(k)$ borders (Lemma 3.3), the dynamic programming table has $O(kn^2)$ entries. We formally define the dynamic programming table below and show how to compute each entry in $O(n^3)$ time.

5.1 Dynamic programming tree

Let $\mathcal{R} = \{R_i\}_{i=1}^k$ and \mathcal{B} be the set of all borders between all pairs of regions. We assume, without loss of generality, that \mathcal{R} is connected, otherwise we split the problem into independent subproblems, one for each connected component of \mathcal{R} .

We define a graph T (that we will argue is a tree) whose edges are the shared borders \mathcal{B} between the regions \mathcal{R} . Two distinct borders $B_{i,j}$ and $B_{h,\ell}$ are incident in this graph if there is an endpoint x of $B_{i,j}$ and y of $B_{h,\ell}$ that are connected by an x -to- y curve in $\mathbb{R}^2 \setminus F$ that does not touch any region \mathcal{R} except at its endpoints x and y ; see Figure 3. Note that this curve may be trivial (i.e. $x = y$). The vertices of T (in a non-degenerate instance) correspond one-to-one with components of $\mathbb{R}^2 \setminus (F \cup \mathcal{R})$ (plus some additional trivial components if three or more regions intersect at a point), or *non-regions*. The edges of T cannot form a cycle, since by the Jordan Curve Theorem this would define a disk that is disjoint from ∂F ; an edge $B_{i,j}$ in the cycle bounds two regions, one of which would be contained by the disk, contradicting that each region shares a boundary with ∂F . Therefore T is indeed a tree. We use an embedding of T that is derived naturally from the embedding of G according to this construction. We use this tree to guide the dynamic program.

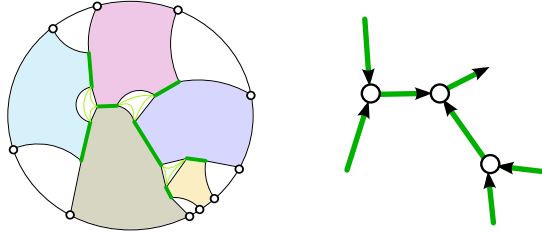


Figure 3: (left) Thick green segments are borders, thin green curves are in $\mathbb{R}^2 \setminus (F \cup \mathcal{R})$ connecting borders that are incident in T . (right) The directed tree T used for dynamic programming.

By the correspondence of the vertices of T to non-regions, we have:

Observation 5.1. *The borders $B_{i,j}, B_{i,\ell}, \dots$ along a given region R_i form a path in T . The order of the borders from s_i to t_i along P_i is the same as in the path in T .*

Consider two edges $B_{i,j}$ and $B_{h,\ell}$ that are incident to the same vertex v of T and that are consecutive in a cyclic order of the edges incident to v in T 's embedding. By Observation 5.1 and the embedding of T , there is a labeling of i, j, h, ℓ such that:

Observation 5.2. *Either $j = h$ or $B_{i,j}$ is the last border of R_j along P_j and $B_{h,\ell}$ is the first border of R_h along P_h .*

Root T at an arbitrary leaf. Since \mathcal{R} is connected, the leaf of T is non-trivial; that is, it has an edge $B_{i,j}$ incident to it. By Observation 5.1, $B_{i,j}$ is (w.l.o.g.) the last border of R_i along P_i and the first border of R_j along P_j . By the correspondence of the vertices of T to non-regions, and the connectivity of \mathcal{R} , either $j = i + 1$ or $i = 1$ and $j = k$. For ease of notation, we assume that the terminals are numbered so that $i = 1$ and $j = k$. We get:

Observation 5.3. *Every non-root leaf edge of T corresponds to a border $B_{i,i+1}$.*

We consider the borders to be both paths in G and edges in T . In T we orient the borders toward the root. In G , this gives a well defined start $a_{i,j}$ and endpoint $b_{i,j}$ of the corresponding path $B_{i,j}$ (note that $a_{i,j} = b_{i,j}$ is possible). By our choice of terminal numbering and orientation of the edges of T , from s_i to t_i along P_i , $b_{i,j}$ is visited before $a_{i,j}$, and from s_j to t_j along P_j , $a_{i,j}$ is visited before $b_{i,j}$.

5.2 Dynamic programming table

We populate a dynamic programming table $D_{i,j}$ for each border $B_{i,j}$. $D_{i,j}$ is indexed by two vertices x and y : x is a vertex of $P_i[b_{i,j}, t_i]$ and y is a vertex of $P_j[s_j, b_{i,j}]$. $D_{i,j}[x, y]$ is defined to be the minimum length of a set of vertex-disjoint paths that connect:

$$x \text{ to } t_i, s_j \text{ to } y, \text{ and } s_h \text{ to } t_h \text{ for every } h = i+1, \dots, j-1$$

These paths are illustrated in Figure 4. We interpret y as the last vertex of $P_j[s_j, b_{i,j}]$ that is used in this sub-solution and we interpret x as the first vertex of $P_i[b_{i,j}, t_i]$ that can be used in this sub-solution (or, more intuitively, the last vertex of the reverse of $P_i[b_{i,j}, t_i]$). By Lemma 3.1, each of the paths defining $D_{i,j}[x, y]$ are contained by their respective region.

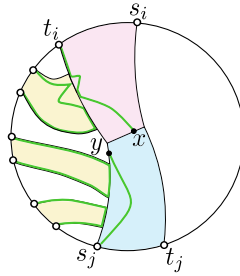


Figure 4: An illustration of the paths defined by $D_{i,j}[x, y]$.

Optimal solution Given $D_{1,k}$, we can compute the value of the optimal solution. By Lemma 3.4, Q_1 and Q_k contain a shortest (possibly trivial) path from s_1 to a vertex x on P_1 , and from a vertex y on P_k to t_k , respectively. Let y be the last vertex of $P_k[s_k, b_{1,k}]$ that Q_k contains and let x be the first vertex of $P_1[b_{1,k}, t_1]$ that Q_1 contains. Then, by Lemma 3.4, the optimal solution has length $D_{1,k}[x, y] + \ell(\alpha(s_1, x)) + \ell(\alpha(y, t_k))$ where α is a Type I path between the given vertices. The optimal solution can be computed in $O(n^2)$ time by enumerating over all choices of x and y . Computing all such Type I paths takes $O(n^2)$ since there are $O(n)$ such paths to compute, and each path can be found using the linear-time shortest paths algorithm for planar graphs [HKRS97].

Base case: Leaf edges of T Consider a non-root leaf edge of T , which, by Observation 5.3, is $B_{i,i+1}$ for some i . Then $D_{i,i+1}[x, y]$ is the length of minimum vertex disjoint x -to- t_i and s_{i+1} -to- y paths in $R_i \cup R_{i+1}$. By Lemma 4.2, $D_{i,i+1}[x, y]$ can be computed in $O(n)$ time for any x and y and so $D_{i,i+1}$ can be populated in $O(n^2)$ time.

5.3 Non-base case of the dynamic program

Consider a border $B_{i,j}$ and consider the edges of T that are children of $B_{i,j}$. These edges considered counter-clockwise around their common node of T correspond to borders $B_{i_1,j_1}, B_{i_2,j_2}, \dots, B_{i_t,j_t}$

where $i \leq i_1 \leq j_1 \leq \dots \leq i_t \leq j_t \leq j$. For simplicity of notation, we additionally let $j_0 = i$ and $i_{t+1} = j$. Then, by Observation 5.2, either $j_\ell = i_{\ell+1}$ or B_{i_ℓ, j_ℓ} is the last border on P_{j_ℓ} and $B_{i_{\ell+1}, j_{\ell+1}}$ is the first border on $P_{i_{\ell+1}}$ for $\ell = 0, \dots, t$.

An acyclic graph H to piece together sub-solutions To populate $D_{i,j}$ we create a directed acyclic graph H with sources corresponding to vertices of $P_i[b_{i,j}, t_i]$ and sinks corresponding to $P_j[s_j, b_{i,j}]$. A source-to-sink (u -to- v) path in H will correspond one-to-one with vertex disjoint paths from:

$$u \text{ to } t_i, s_j \text{ to } v, \text{ and } s_h \text{ to } t_h \text{ for every } h = i+1, \dots, j-1$$

Here u and v do not correspond to the vertices x and y that index $D_{i,j}$; to these vertex disjoint paths, we will need to append vertex disjoint x -to- u and v -to- y paths (which can be found using a minimum cost flow computation by Lemma 4.2).

The arcs of H are of two types: (a) Type I arcs and (b) sub-problem arcs. Directed paths in H alternate between these two types of arcs. The Type I arcs correspond to Type I paths and the endpoints of the Type I arcs correspond to the endpoints of the Type I paths. Sub-problem arcs correspond to the sub-solutions from the dynamic programming table and the endpoints of the sub-problem arcs correspond to the indices of the dynamic programming table (and so are the endpoints of the *incomplete* paths represented by the table). Note that vertices of a border may appear as either the first or second index to the dynamic programming table; in H , two copies of the border vertices are included so the endpoints of the resulting sub-solution arcs are distinct. Formally:

- Type I arcs go from vertices of P_{j_ℓ} to vertices of $P_{i_{\ell+1}}$ for $\ell = 0, \dots, t$. Consider regions R_{j_ℓ} and $R_{i_{\ell+1}}$. There are two cases depending on whether or not $R_{j_\ell} = R_{i_{\ell+1}}$.
 - If $R_{j_\ell} = R_{i_{\ell+1}}$, then for every vertex x of $P_{j_\ell}[s_{j_\ell}, b_{i_\ell, j_\ell}]$ and every vertex y of $P_{j_\ell}[b_{i_{\ell+1}, j_{\ell+1}}, t_{j_\ell}]$, we define a Type I arc from x to y with length equal to the length of the x -to- y Type I path.
 - If $R_{j_\ell} \neq R_{i_{\ell+1}}$, then for every vertex x of $P_{j_\ell}[s_{j_\ell}, b_{i_\ell, j_\ell}]$ and every vertex y of $P_{i_{\ell+1}}[b_{i_{\ell+1}, j_{\ell+1}}, t_{j_\ell}]$, we define a Type I arc from x to y with length equal to the sum of the lengths of the x -to- t_{j_ℓ} and $s_{i_{\ell+1}}$ -to- y Type I paths.
- Sub-problem arcs go from vertices of P_{i_ℓ} to vertices of P_{j_ℓ} for $\ell = 1, \dots, t$. For every $\ell = 1, \dots, t$ and every vertex x of P_{i_ℓ} and vertex y of P_{j_ℓ} (that are not duplicates of each other), we define a sub-problem arc from x to y with length equal to $D_{i_\ell, j_\ell}[x, y]$.

Shortest paths in H By construction of H and the definition of D_{i_ℓ, j_ℓ} , for a source u and sink v , we have:

Observation 5.4. *There is a u -to- v path in H with length L if and only if there are vertex disjoint paths of total length L from u to t_i , s_j to v , and s_h to t_h for every $h = i+1, \dots, j-1$.*

See Figure 5 for an illustration of the paths in G that correspond to a source-to-sink path in H . Let $H[u, v]$ denote the shortest u -to- v path in H (for a source u and a sink v). We will need to compute $H[u, v]$ for every pair of sources and sinks. Since every vertex in G appears at most twice in H , the size of H is $O(n^2)$ and for a given sink and for all sources, the shortest source-to-sink paths can be found in time linear in the size of H using dynamic programming. Repeating for all sinks results in an $O(n^3)$ running time to compute $H[u, v]$ for every pair of sources and sinks.²

²Computing the length of the Type I paths is dominated by $O(n^3)$, but can be improved to $O(n \log n)$ time by

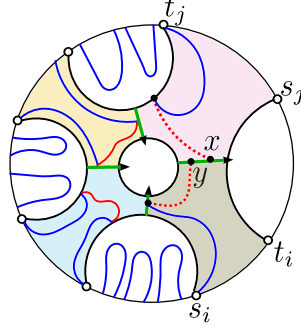


Figure 5: Mutually disjoint walks represented by a directed path in H for a set of incident borders (green). The blue arcs correspond to the walks represented in sub-problems and the solid red paths correspond to the Type I paths represented by Type I arcs. The dotted red paths represent vertex disjoint u -to- x and v -to- y paths that will be added via a min-cost flow computation.

Handling vertices that appear in more than two regions As indicated, a vertex c may appear in more than two regions; this occurs when two or more borders share an endpoint. In the construction above, if c appears in only two regions, then, c can only be used as the endpoint of two sub-paths (whose endpoints meet to form a part of an s_i -to- t_i path in the global solution). However, suppose for example that c appears as an endpoint of both $B_{i,j}$ and $B_{i',j'}$ and so 4 copies of c are included in H (two copies for each of these borders). On the other hand, one need only *guess* which s_i -to- t_i path c should belong to first and construct H accordingly. There are only k possibilities to try.

Unfortunately, there may be $O(k)$ shared vertices among the borders $B_{i_1,j_2}, B_{i_2,j_2}, \dots, B_{i_t,j_t}$ involved in populating $D_{i,j}$. It seems that for each of these $O(k)$ shared vertices, one would need to guess which s_i -to- t_i path it belongs to, resulting in an exponential dependence on k .

Here we recall the structure of T : the nodes of T correspond to *non-regions*: disks (or points) surrounded by regions. If there are several shared vertices among the borders, then there is an order of these vertices around the boundary of the non-region. That is, for a vertex shared by a set of borders, these borders must be contiguous subsets of $B_{i_1,j_2}, B_{i_2,j_2}, \dots, B_{i_t,j_t}$. In terms of the construction of H , there is a contiguous set of levels that a given shared vertex appears in and distinct shared vertices participate in non-overlapping sets of levels. For one set of these levels, we can create different copies of the corresponding section of H . In each copy we modify the directed graph to reflect which s_i -to- t_i path the corresponding shared vertex may belong to (see Figure 6). As we have argued, since distinct shared vertices participate in non-overlapping sets of levels, this may safely be repeated for every shared vertex. The resulting graph has size $O(kn^2)$ since there are $O(k)$ borders and shared vertices are shared by borders. The resulting running time for computing all source-to-sink shortest paths in the resulting graph is then $O(kn^3)$.

Computing $D_{i,j}$ from H To compute $D_{i,j}[x, y]$, we consider all possible u on $P_i[x, t_i]$ and v on $P_j[s_j, y]$ and compute the minimum-length vertex disjoint u -to- x path and v -to- y path that only use vertices that are interior to $R_i \cup R_j$ (that is vertices of $B_{i,j}$ may be used); by Lemma 4.2, these paths can be computed in linear time. Let $M[u, v]$ be the cost of these paths. Then

$$D_{i,j}[x, y] = \min_{u \in P_i[x, t_i], v \in P_j[s_j, y]} M[u, v] + H[u, v].$$

running Klein's boundary shortest path algorithm [Kle05] in all regions, resulting in an $O(n^2)$ time to construct H .

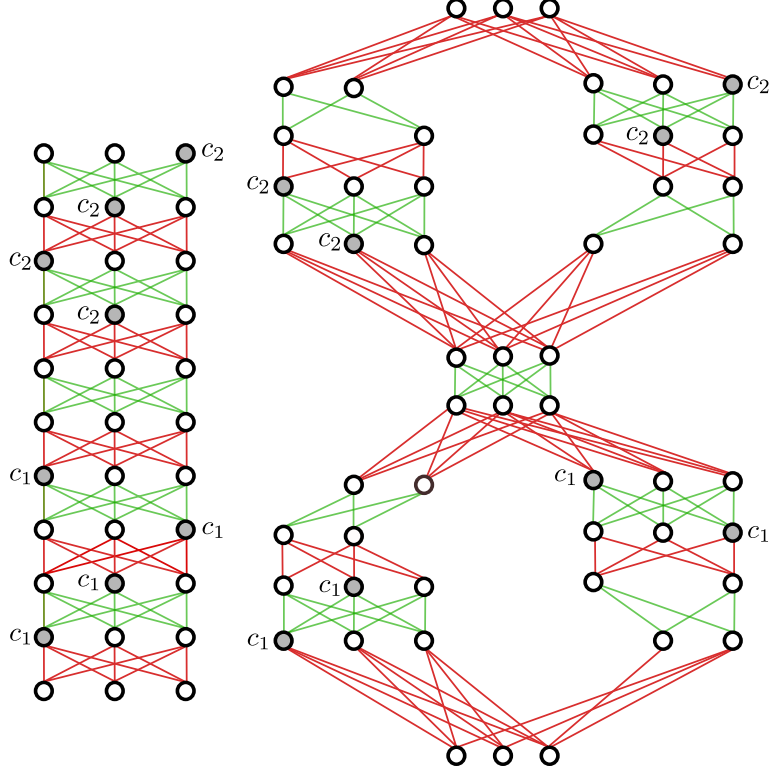


Figure 6: (left) H constructed without handling the fact that vertices c_1 and c_2 (black) may appear more than twice. The arcs are all directed upwards (arrows are not shown); green arcs are sub-problem arcs and red arcs are Type I arcs. (right) The levels that c_1 appears in are duplicated, and only one pair of copies of c is kept in each copy of these levels. The vertex c_1 may only be visited twice now on a source to sink path.

As there are $O(n^2)$ choices for u and v and $M[u, v]$ can be computed in linear time, $D_{i,j}[x, y]$ can be computed in $O(n^3)$ time given that distances in H have been computed.

Overall running time For each border $B_{i,j}$, H is constructed and shortest source-to-sink paths are computed in $O(kn^3)$ time. For each $x, y \in B_{i,j}$, $D_{i,j}[x, y]$ is computed in $O(n^3)$ time. Since there are $O(n^2)$ pairs of vertices in $B_{i,j}$, $D_{i,j}$ is computed in $O(n^5)$ time (dominating the time to construct and compute shortest paths in H). Since there are $O(k)$ borders (Lemma 3.3), the overall time for the dynamic program is $O(kn^5)$.

Acknowledgements This material is based upon work supported by the National Science Foundation under Grant No. CCF-1252833.

References

- [BH14] Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *Automata, Languages, and Programming*, pages 211–222. Springer, 2014.
- [CMPP13] M. Cygan, D. Marx, M. Pilipczuk, and M. Pilipczuk. The planar directed k-vertex-disjoint paths problem is fixed-parameter tractable. In *Proceedings of the 2013 IEEE*

54th Annual Symposium on Foundations of Computer Science, FOCS '13, pages 197–206, Washington, DC, USA, 2013. IEEE Computer Society.

- [FHW80] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111 – 121, 1980.
- [HKRS97] Monika R. Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, August 1997.
- [Kar74] Richard Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1974.
- [Kle05] Philip N. Klein. Multiple-source shortest paths in planar graphs. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 146–155, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
- [KS10] Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010.
- [KvL84] MR Kramer and Jan van Leeuwen. The complexity of wire-routing and finding minimum area layouts for arbitrary vlsi circuits. *Advances in computing research*, 2:129–146, 1984.
- [KW10] Ken-ichi Kawarabayashi and Paul Wollan. A shorter proof of the graph minor algorithm: The unique linkage theorem. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 687–694, New York, NY, USA, 2010. ACM.
- [MP93] Matthias Middendorf and Frank Pfeiffer. On the complexity of the disjoint paths problem. *Combinatorica*, 13(1):97–107, 1993.
- [MVV87] K. Mulmuley, V. Vazirani, and U. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):345–354, 1987.
- [ORS93] Richard G Ogier, Vladislav Rutenburg, and Nachum Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *Information Theory, IEEE Transactions on*, 39(2):443–455, 1993.
- [RS95] Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.
- [Sch94] Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal on Computing*, 23(4):780–788, 1994.
- [SM05] Anand Srinivas and Eytan Modiano. Finding minimum energy disjoint paths in wireless ad-hoc networks. *Wireless Networks*, 11(4):401–417, 2005.
- [VS11] Éric Colin De Verdière and Alexander Schrijver. Shortest vertex-disjoint two-face paths in planar graphs. *ACM Transactions on Algorithms (TALG)*, 7(2):19, 2011.